PATTERSON & SHERIDAN, LLP
ATTORNEYS AT LAW
ATTORNEYS AT LAW

3040 Post Oak Blvd, Suite 1500
Houston, TX 77056-6582
TEL 713.623.4844
FAX 713.623.4846

FACSIMILE COVER SHEET

DATE: November 5, 2008

FILE NO: ROC920000051US2 (1032.012185 (IBM2K0051.D1))

TO: Examiner Ben C. Wang

FAX NO: 1-571-270-2240

2nd Interview
Q2-07

FROM: Gero G. McClellan / Johnny Lam

PAGE(S) with cover: 4

RE:

TITLE: DEBUGGING METHODS FOR HEAP MISUSE
U.S. SERIAL NO.: 10/661,982
FILING DATE: September 12, 2003
INVENTOR(S): Bates et al.
EXAMINER: Ben C. Wang
GROUP ART UNIT: 2192
CONFIRMATION NO.: 9327

Attached are the following document(s) for the above-referenced application:

- Agenda for Interview Request

From: Gero G. McClellan, Reg. No. 44227
Contact: Johnny Lam, (336) 698-4288

To: **Examiner Ben Wang**
Fax: (571) 270-2240
Re: Agenda for Interview Request
Attorney Docket Number: IBM2K0051.D1

Application No.: 10/661,982
First Named Applicant: Cary Lee Bates
Art Unit: 2192
Status of Application: Pending

Agenda for Interview Request – UNOFFICIAL: DO NOT ENTER

In this case, *Spertus* does not disclose "each and every element as set forth in the claim." For example, *Spertus* does not disclose "allowing a user to establish, via the computer user interface, a relationship between one or more of the memory deallocators and one or more of the memory allocators, wherein the relationship requires that memory space allocated by the one or more allocators is freed by the one or more deallocators." The Examiner suggests that this limitation is disclosed by *Spertus*, Fig. 9; col. 15, lines 20-42. Specifically, the Examiner asserts as follows:

> As to claim 1, Spertus discloses . . . allowing a user to establish, via the computer interface, a relationship between one or more of the memory deallocators and one or more of the memory allocators, wherein the relationship requires that memory space allocated by the one or more allocators is freed by the one or more deallocators (e.g., Fig. 9; Col. 15, Lines 20-42 – Fig. 9 shows the structure of a bucket 901. Each bucket 901 is made up of a page information structure 907 which includes information 904 indicating the size of the objects that will be allocated from the bucket and a free list pointer 906 indicating the head of the list of unallocated object 911 contained in bucket 901 ...) . . . .

Office Action, page 3 (emphasis original). However, the cited portions of *Spertus*, and in fact *Spertus* as a whole, fail to disclose anything at all about "a relationship between one or more of the memory deallocators and one or more of the memory allocators." The cited portions, as well as Figure 9, of *Spertus* are reproduced below:

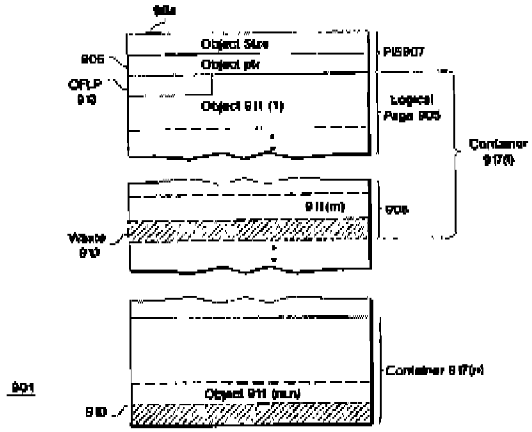

Fig. 9

> FIG. 9 shows the structure of a bucket 901. Each bucket 901 is made up of a page information structure 907 which includes information 904 indicating the size of the objects that will be allocated from the bucket and a free list pointer 906 indicating the head of the list of unallocated objects 911 contained in bucket 901. The objects 911 are stored in one or more containers 917, which are made up of one or more contiguous logical pages 905. The size of the container is the number of logical pages 905 which reduces waste 910 to a minimum for the size of object 911 stored in the bucket. Each object 911(i) which has not yet been allocated has an object free list pointer 913 pointing to the next unallocated object 911. When an object is allocated, the free list pointer 913 is overwritten by data and free list pointer 906 is set to point to the next unallocated object 911 in the list. When an object is freed, the allocator to which the bucket belongs links the freed object in at the head of the free list. This has the advantage that an object has been used tends to be reused before it is removed from the caches in the machine that the program that uses the object is executing on. When a bucket 901 runs out of unallocated objects 911, the allocator that uses the bucket obtains an additional container 917 for the bucket from the large object allocator.

Spertus, Fig 9; col. 15, lines 20-42. In other words, the Examiner cites a portion of *Spertus* that discusses the structure of a bucket used to facilitate heap defragmentation. Such a discussion about heap defragmentation, and in fact *Spertus* as a whole, fails to disclose anything at all about a relationship between a memory deallocator and a memory allocator. In particular, *Spertus* makes no reference of any kind to establishing relationships between deallocators and allocators. In fact, a simple word search of *Spertus* reveals no instances of the words "relationship" or "deallocator". Thus, contrary to the Examiner's suggestion, *Spertus* does not disclose "a relationship between one or more of the memory deallocators and one or more of the memory allocators." Accordingly, Applicants submit that the rejection is defective and should be withdrawn.

Further, for the reasons set forth above, it also follows that *Spertus* does not disclose "upon a call to the one or more deallocators to free a memory space, determining whether the relationship is violated." The Examiner suggests that Spertus discloses this limitation in *Spertus*, Fig. 8; col. 2, lines 59-62; and col. 5; lines 51-67. Specifically, the Examiner asserts as follows:

> allowing the code to execute; upon a call to the one or more deallocators to free a memory space, determining whether the relationship is violated (e.g., Fig. 8, element 811 - gcLogAllLeaks - ... write all leaks to the log; Col. 2, Lines 59-62 ... provides memory debugging information such as memory allocations, memory leaks, and current heap size; Col. 5, Lines 51-67 - ... One problem detected by a memory debugger is memory 'leaks' ...); and

Office Action, page 3. In other words, the Examiner cites portions of *Spertus* that discuss memory debugging information, such as memory allocations and memory leaks. Such a discussion about memory allocations and memory leaks fails to disclose anything at all about "determining whether a relationship [between a deallocator and an allocator] is violated." Even more problematic is the Examiner's suggestion that a discussion of memory leaks teaches "<u>upon a call</u> to the one or more deallocators." To illustrate, *Spertus* describes a memory leak as follows:

> One problem detected by a memory debugger is memory "leaks", which occur when a program contains code that allocates memory, <u>but does not contain code that frees the allocated memory</u> when it is no longer being used by the program. Leaks of course always waste memory; with serious leaks, all of the heap memory available to the program may be occupied by leaks, and in that case, the program will fail when a new allocation is attempted.

Spertus, col. 5, lines 57-65 (emphasis added). Put another way, memory leaks only occur when <u>a deallocator is never called</u>, even when memory is no longer being used. Therefore, the Examiner's suggestion that "when a deallocator is never called" teaches "upon a call to the one or more deallocators" is wholly contradictory and utterly untenable. Accordingly, Applicants submit that the rejection is defective and should be withdrawn.